과제 #1
<총 16점>

- 총 3 문제: Q1(2 점), Q2 (4 점), Q3 (10 점)
- 제출기한: 2022년 11월 14일 24:00 까지
- 제출물: MS VS 솔루션 폴더 (압축파일) 또는 소스 (압축) 파일

※ 충분한 주석은 필수입니다.

※ **제출 파일에 소스가 누락되지 않게 주의하세요.**

[Q1] 밑 빠진 스택 [2 점]

- Drop-Out Stack은 일종의 밑 빠진 스택입니다. 제한된 크기의 순차 구조로 구현된 스택의 일종으로, 포화 상태인 경우에도 Push 연산은 오류로 처리하지 않고, 스택의 바닥에 있는 데이터를 삭제하고, 저장되어 있는 데이터를 아래로 이동 후, 정상적으로 수행됩니다. 스택 공간이 제한되어 있을 때, 새로운 물건을 쌓기 위해 가장 오래 동안 쌓여 있던 물건을 차례로 없애는 방법이기도 합니다. 이러한 특성을 가진 스택을 배열로 구현하고, 아래 사항을 수행하는 C 프로그램을 작성합니다. (교재 예제 5-1 참조)
  1) 스택의 크기를 10 이하의 자연수 N으로 입력 받습니다.
  2) 데이터, 문자 'A' 부터 'K'까지 하나씩 스택에 Push합니다.
  3) 스택에 저장되어 있는 모든 데이터-문자를 Pop하여 순서대로 출력합니다. (아래의 입/출력 예시 참조)
  4) 상기 1) ~ 3) 과정을 반복 수행합니다. N = 0인 경우 종료합니다.

※ 입/출력 예시

```
*** 스택 크기 (10 이하 자연수), N = 1
Chars in Stack [1]:  K
*** 스택 크기 (10 이하 자연수), N = 3
Chars in Stack [3]:  K  J  I
*** 스택 크기 (10 이하 자연수), N = 5
Chars in Stack [5]:  K  J  I  H  G
*** 스택 크기 (10 이하 자연수), N = 7
Chars in Stack [7]:  K  J  I  H  G  F  E
*** 스택 크기 (10 이하 자연수), N = 11
+++ 10 이하 자연수를 입력하세요!
*** 스택 크기 (10 이하 자연수), N = 9
Chars in Stack [9]:  K  J  I  H  G  F  E  D  C
*** 스택 크기 (10 이하 자연수), N = 0
Bye!!!
```

[교재 예제 5-1]
순차 자료구조를 이용해 순차 스택 구현하기 프로그램

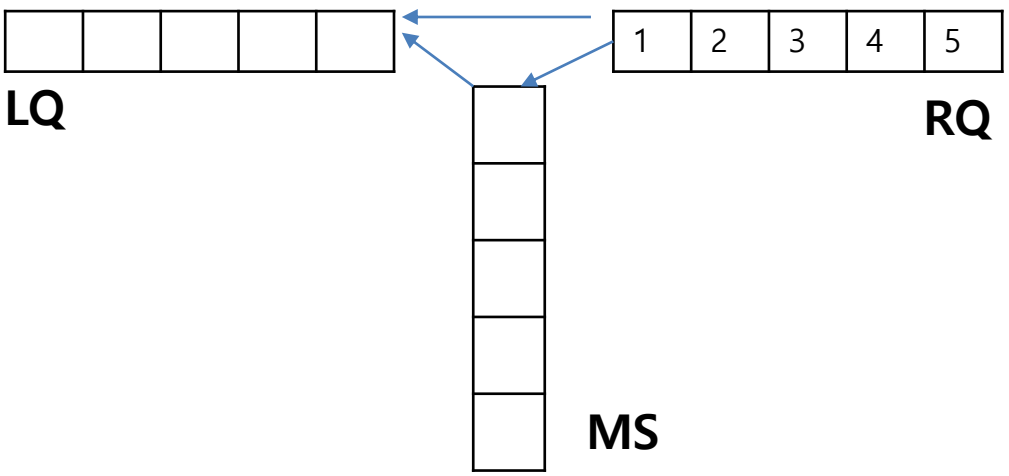
```c
#include <stdio.h>
#include "stackS.h"

int top = -1;

// 스택이 공백 상태인지 확인하는 연산
int isStackEmpty() {
	if (top == -1) return 1;
	else return 0;
}

// 스택이 포화 상태인지 확인하는 연산
int isStackFull() {
	if (top == STACK_SIZE - 1) return 1;
	else return 0;
}

// 스택의 top에 원소를 삽입하는 연산
void push(element item) {
	if (isStackFull()) {// 스택이 포화 상태인 경우
		printf("₩n₩n Stack is FULL! ₩n");
		return;
	} else stack[++top] = item;// top을 증가시킨 후 현재 top에 원소 삽입
}

// 스택의 top에서 원소를 삭제하는 연산
element pop() {
	if (isStackEmpty()) {// 스택이 공백 상태인 경우
		printf("₩n₩n Stack is Empty!!₩n");
		return 0;
	} else return stack[top--];// 현재 top의 원소를 삭제한 후 top 감소
}

// 스택의 top 원소를 검색하는 연산
element peek() {
	if (isStackEmpty()) {// 스택이 공백 상태인 경우
		printf("₩n₩n Stack is Empty !₩n");
		exit(1);
	} else return stack[top];// 현재 top의 원소 확인
}

// 스택의 원소를 출력하는 연산
void printStack() {
	int i;
	printf("₩n STACK [ ");
	for (i = 0; i <= top; i++)
	printf("%d ", stack[i]);
	printf("] ");
}
```

```c
#pragma once
#define STACK_SIZE 100

typedef int element;		// 스택 원소(element)의 자료형을 int로 정의
element stack[STACK_SIZE];	// 1차원 배열 스택 선언

int isStackEmpty();
int isStackFull();
void push(element item);
element pop();
element peek();
void printStack();
```

[Q2] 스택 수 [4 점]

- 교재 6장 응용 예제 08, '스택 수 판별하기'를 토대로 일반화한 문제입니다. 오른 쪽 큐-RQ에 1에 N까지의 자연수를 순서대로 넣고, 하나씩 꺼내, 중간 스택-MS를 거치거나 거치지 않고, 왼쪽 큐-LQ로 이동시켰을 때 만들어 질 수 있는 모든 **스택 수**를 출력하는 C 프로그램을 작성/구현합니다.
  1) 5이하의 자연수 N을 입력 받아, 1에서 N까지의 자연수를 순서대로 RQ에 넣습니다.
  2) LQ에 나타날 수 있는 가능한 모든 스택 수를 출력하고, 출력한 총 수택 수의 개수도 같이 출력합니다. (다음 쪽의 입/출력 예시 참조)
  3) 상기 1), 2) 과정을 반복 수행합니다. N = 0인 경우 종료합니다.

※ N = 5일 경우, 초기화 구성 예



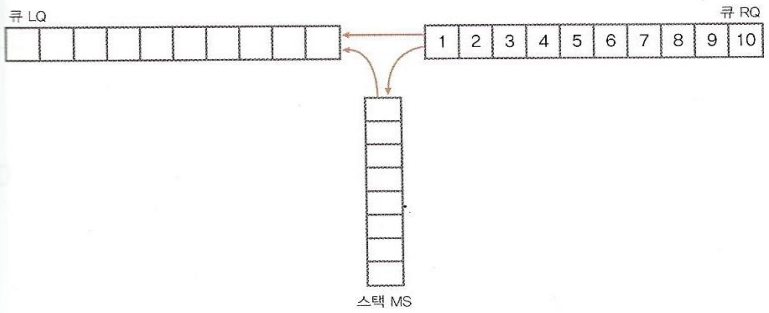
※ 입/출력 예시

```
*** 숫자 수 (5 이하 자연수), N = 1
   1: 1
  # of Stack Numbers = 1
*** 숫자 수 (5 이하 자연수), N = 2
   1: 12
   2: 21
  # of Stack Numbers = 2
*** 숫자 수 (5 이하 자연수), N = 3
   1: 123
   2: 132
   3: 231
   4: 213
   5: 321
  # of Stack Numbers = 5
*** 숫자 수 (5 이하 자연수), N = 4
   1: 1234
   2: 1243
   3: 1342
   4: 1324
   5: 1432
   6: 2341
   7: 2314
   8: 2431
   9: 2134
  10: 2143
  11: 3421
  12: 3241
  13: 3214
  14: 4321
  # of Stack Numbers = 14
*** 숫자 수 (5 이하 자연수), N = 6
+++ 5 이하 자연수를 입력하세요!
*** 숫자 수 (5 이하 자연수), N = 0
Bye!!!
```

응용예제
08

스택 수 판별하기

스택 수Stack Number란 아래의 장치를 가지고 왼쪽 큐 LQ에 만들 수 있는 수열을 말한다. 아래의 장치는 두 개의 큐와 한 개의 스택으로 구성되어 있는데 오른쪽 큐 RQ에는 1부터 N까지 정수가 순서대로 들어 있다. 오른쪽 큐에서 데이터를 하나씩 꺼내서 왼쪽 큐에 넣거나 중간에 있는 스택에 넣을 수 있다. 즉, 왼쪽 큐에 들어가는 숫자는 오른쪽 큐에서 직접 입력되거나 스택에서 pop()하여 꺼낸 데이터다. 왼쪽 큐에서 오른쪽 큐로 데이터를 넣거나 스택에서 오른쪽 큐로 데이터를 넣을 수는 없다.



문제 **스택 수 판단하기**

스택 수를 생성하는 프로그램을 작성하여 입력된 수열이 스택 수인지를 판별하여 출력하시오.

① 입력 조건
- 첫째 줄에 오른쪽 큐에 들어가는 데이터 크기 N의 정수가 주어진다. ($3 \le N \le 20$)
- 둘째 줄에 테스트할 수열이 N개의 숫자로 주어진다.

② 출력 조건
- 첫째 줄에 테스트 결과가 스택 수이면 POSSIBLE을 출력하고 스택 수가 아니면 IMPOSSIBLE을 출력한다.

③ 입출력 예시

```
10
3 4 6 7 8 5 9 2 1 10
```

→

```
POSSIBLE
```

문제 해결

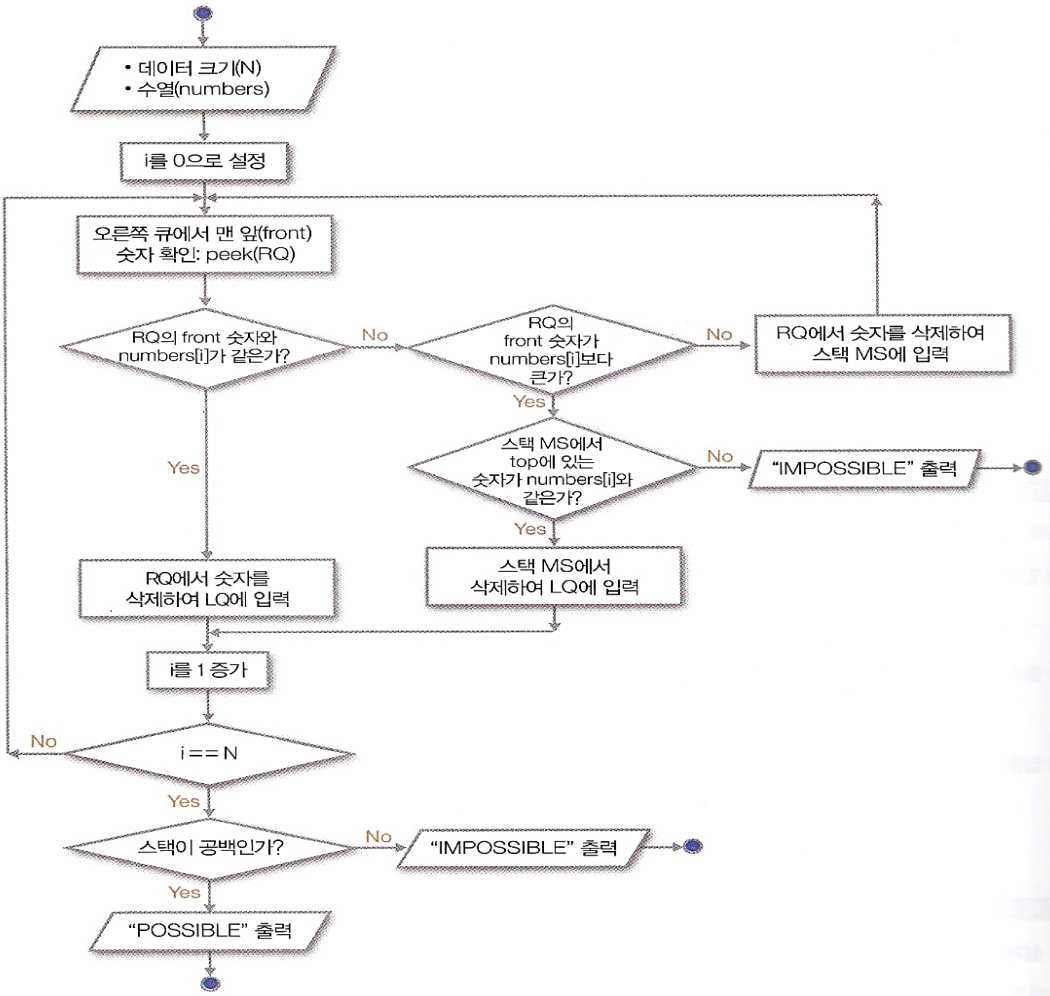
① 설계

스택의 push, pop 연산을 사용한 후입선출 LIFO 동작과 큐의 enQueue, deQueue 연산을 사용한

선입선출 FIFO 동작을 조합하여 해결한다.

- 오른쪽 큐에서 deQueue로 처리할 숫자(현재 큐에서 맨 앞에 있는 숫자)와 테스트할 수열의 숫자가 같으면 오른쪽 큐에서 꺼내서(deQueue) 왼쪽 큐로 직접 입력(enQueue)한다.
- 그렇지 않고 오른쪽 큐의 숫자가 테스트할 수열의 숫자보다 작으면 테스트할 수열의 숫자가 나올 때까지 오른쪽 큐의 숫자를 스택에 입력(push)한다.
- 또는 오른쪽 큐의 숫자가 테스트할 수열의 숫자보다 크면 스택에서 pop하여 왼쪽 큐에 입력한다.
- 이 규칙으로 만들어지지 않는 수열은 스택 수가 될 수 없다.

② 순서도



[Q3] 수식 이진 트리 [10점]

- 중위표기 수식을 입력 받아, 아래의 사항을 C 프로그램으로 작성/구현합니다.
  - ① 이진 트리를 **자동 생성**합니다., (강의자료 -"수식의 이진트리 표현" 참조) <**4**점>
  - ② **레벨 순회** (Breadth-First Traversal) 결과를 레벨 별 노드의 문자와 함께, 이 트리를 포화 이진 트리의 배열로 저장할 때 사용되는 **노드 번호**를 같이 출력합니다. <**4**점>
  - ③ **후위 순회**를 수행하여, 순회 과정과 함께 수식의 **연산 결과 값**을 출력합니다. <**2**점>
- 처리할 각 중위 수식은 첨부된 "InputInfix.txt"에 한 줄씩 표현되어 있습니다.
- 다음 쪽 출력 예시를 참조하여 출력 코드를 작성합니다.

※ ①번 알고리즘에 대한 설명은 필수 주석입니다.

※ 스택, 큐, 이진 트리 등에 대한 종합적인 복습 과제입니다. 그 동안 학습한 예제 소스코드들을 충분히 이해하고 최대한 활용하기 바랍니다.

※ 제한 사항

- 중위표기수식은 피연산자, 연산자, 왼쪽 소괄호, 오른쪽 소괄호로 구성된다.
- 중위표기수식은 Newline('**₩n**') 문자로 끝난다. 즉, 수식은 한 줄로 표현된다.
- 피연산자는 자연수 한 자리 숫자로, 연산자는 2항산술연산자('**+**', '**-**', '*****', '**/**', '**%**'), 그리고 소괄호('**(**', '**)**') 등으로 수식을 표현한다.
- 연산자 우선순위와 결합법칙
  - ✓ 연산자 중, 곱하기(*), 나누기(/), 나머지(%) 등은 같은 우선 순위를 가지며,, 2항연산자 더하기(+)와 빼기(-)도 같은 우선순위를 가진다. 모든 연산자는 왼쪽에서 오른쪽으로 결합법칙이 적용된다.. (1항 연산자, +와 -는 사용하지 않는다.)
  - ✓ 왼쪽 소괄호는 처음엔 (※스택에 들어 오기 전) 모든 연산자보다 높은 우선순위를 가지지만, 짝이 되는 오른쪽 소괄호를 만날 때까지는 (※스택 안에서) 가장 낮은 우선순위를 가진다.
  - ✓ 곱하기(*), 나누기(/), 나머지(%) 등은 더하기(+)와 빼기(-)보다 높은 우선순위를 가진다.

※ 출력 예시

```
********** Tree For Expression **********
* Input Infix String: 1 + (2 - 3 * 4 / (5 + 6)) - 7
  +++ Build Tree +++
  ** Level-order Traversal:
        Leval[ 0]:      -[ 1]
        Leval[ 1]:      +[ 2]   7[ 3]
        Leval[ 2]:      1[ 4]   -[ 5]
        Leval[ 3]:      2[10]   /[11]
        Leval[ 4]:      *[22]   +[23]
        Leval[ 5]:      3[44]   4[45]   5[46]   6[47]
  ** Postorder Traversal:       1234*56+/-+7-   (=> -5)

* Input Infix String: 2 * 5 + 4 % 7
  +++ Build Tree +++
  ** Level-order Traversal:
        Leval[ 0]:      +[ 1]
        Leval[ 1]:      *[ 2]   %[ 3]
        Leval[ 2]:      2[ 4]   5[ 5]   4[ 6]   7[ 7]
  ** Postorder Traversal:       25*47%+ (=> 14)

* Input Infix String: 9 / 8 * 7 * 6 + 5
  +++ Build Tree +++
  ** Level-order Traversal:
        Leval[ 0]:      +[ 1]
        Leval[ 1]:      *[ 2]   5[ 3]
        Leval[ 2]:      *[ 4]   6[ 5]
        Leval[ 3]:      /[ 8]   7[ 9]
        Leval[ 4]:      9[16]   8[17]
  ** Postorder Traversal:       98/7*6*5+       (=> 47)

* Bye
```

※ 포화 이진 트리의 노드 번호

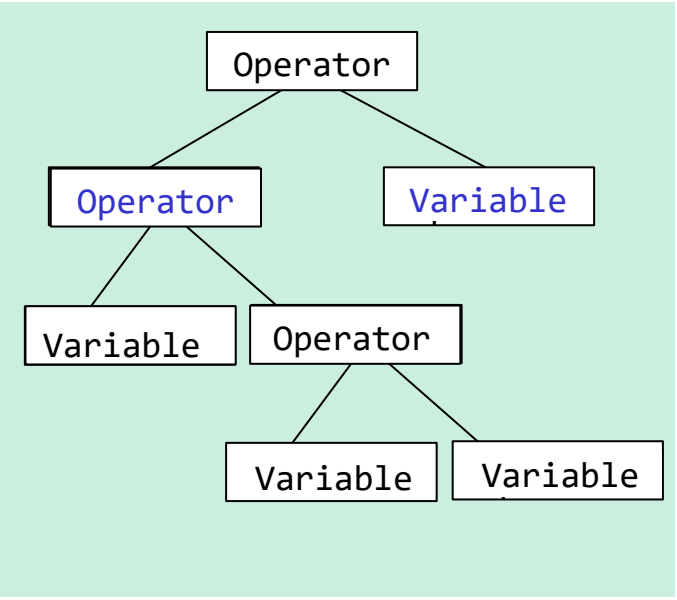
※ **InputInfix.txt** : 중위표기 수식 (3개)

1 + (2 - 3 * 4 / (5 + 6)) - 7
2 * 5 + 4 % 7
9 / 8 * 7 * 6 + 5

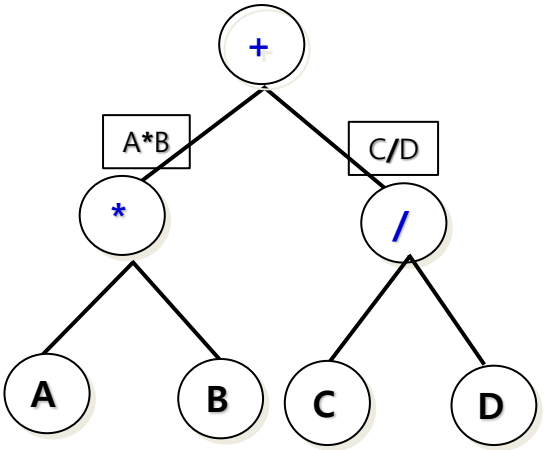
- **수식의 이진 트리 표현**

```
Expression ::= Expression  Operator  Expression |
               Operator Expression |
               Variable
```

※ **B**ackus-**N**aur **F**orm



A * B + C / D



+ 후위 표기법 변환 – 연산자 우선순위 적용

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_STACK_SIZE 100  /* maximum stack size */
#define MAX_EXPR_SIZE 100   /* max size of expression */
typedef enum {  //index to precedence
    lparen, rparen, plus, minus, times, divide,
    mod, eos, operand
} precedence;
typedef precedence element;// 스택 원소(element)의 자료형을 정의
element stack[MAX_STACK_SIZE];  /* global stack */
int top = -1; // stack top index
/* isp and icp arrays - index is value of precedence
   lparen, rparen, plus, minus, times, divide, mod, eos
*/
/* isp: in stack precedence, icp: incoming precedence */
static int isp[] = { 0, 19, 12, 12, 13, 13, 13, 0 };
static int icp[] = { 20, 19, 12, 12, 13, 13, 13, 0 };
char sym[10] = { '(', ')', '+','-', '*', '/', '%',';', '₩0'};

precedence getToken(char symbol) {
    static int  n = 0;
    switch (symbol) {
        case '(': return lparen;
        case ')': return rparen;
        case '+': return plus;
        case '-': return minus;
        case '/': return divide;
        case '*': return times;
        case '%': return mod;
        case '; ': return eos;
        default: return operand;
    }
}
```

```
/*** Infix to Postfix Implementation ***/
void postfix(char in[], char post[]) {
    char symbol; precedence token;
    int i=0, j=0, len = strlen(in);

    push(eos);  // for safety:precedence 0
    for (i = 0; i<len; i++) {
        symbol = in[i];
        token = getToken(symbol);
        if (token == eos) break;
        if (token == operand) post[j++] = symbol;
        else if (token == rparen) {
            while (peek() != lparen) post[j++] = sym[pop()];
            pop(); /* discard the left parenthesis */
        } else {
            while (isp[peek()] >= icp[token])
                    post[j++] = sym[pop()];
            push(token);
        }
    }
    while ((token = pop()) != eos) post[j++] = sym[token];
    post[j++] = '₩0'; // end of string
}
```

```
void main() {
    char strInfix[MAX_EXPR_SIZE] = "9*(1+2)%3;";
    char strPostfix[MAX_EXPR_SIZE];
    printf( "infix: %s₩n", strInfix);
    postfix(strInfix, strPostfix);
    printf("Postfix: %s₩n", strPostfix);
}
```

infix: 9*(1+2)%3;
Postfix: 912+*3%

Prefix: %*9+123

※ **Infix to Prefix**
① **역순으로 Scan**
② **피연산자 Swap**
③ **연산?**

[교재 예제 5-4]

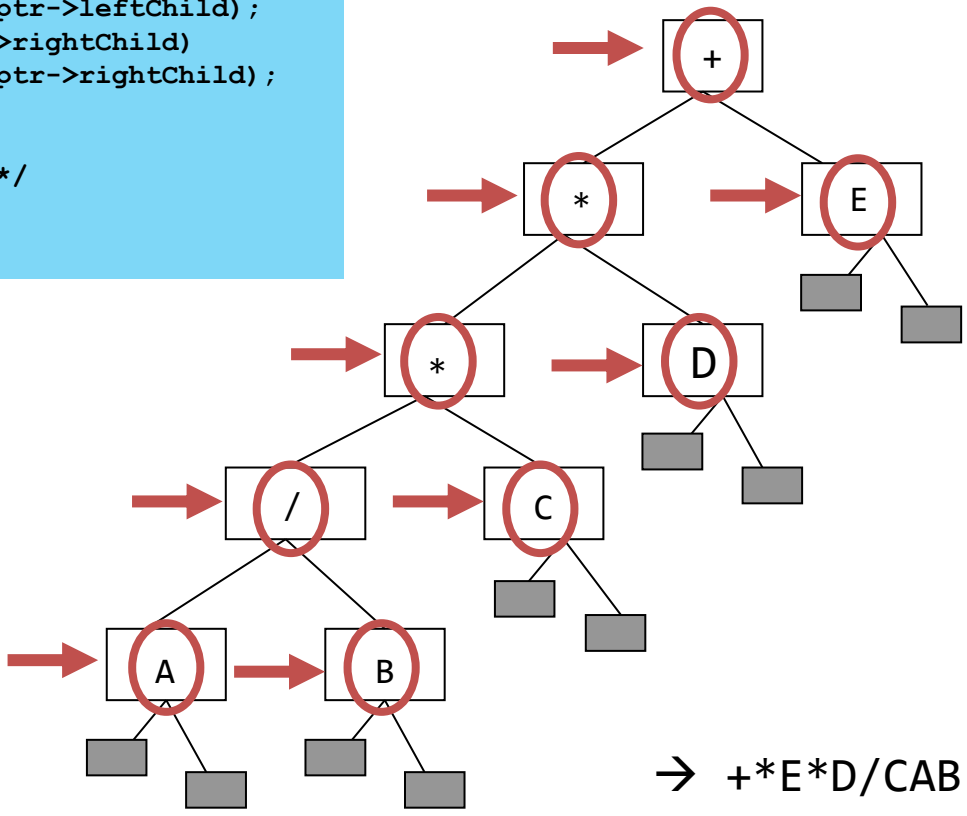
수식을 후위 표기법으로 연산하기 프로그램

```
#include <string.h>
#include "stackL.h"
#include "evalPostfix.h"

// 후위 표기법 수식을 계산하는 연산
element evalPostfix(char* exp) {
	int opr1, opr2, value, i = 0;
	// char형 포인터 매개변수로 받은 수식 exp의 길이를 계산하여 length 변수에 저장
	int length = strlen(exp);
	char symbol;

	top = NULL;

	for (i = 0; i < length; i++) {
		symbol = exp[i];
		if (symbol != '+' && symbol != '-' && symbol != '*' && symbol != '/') {
			value = symbol - '0';
			push(value);
		} else {
			opr2 = pop();
			opr1 = pop();
			// 변수 opr1과 opr2에 대해 symbol에 저장된 연산자를 연산
			switch (symbol) {
				case '+': push(opr1 + opr2); break;
				case '-': push(opr1 - opr2); break;
				case '*': push(opr1 * opr2); break;
				case '/': push(opr1 / opr2); break;
			}
		}
	}
	// 수식 exp에 대한 처리를 마친 후 스택에 남아 있는 결과값을 pop하여 반환
	return pop();
}
```

[강의자료 7장 쪽 48]

+ Level-order Traversal-"Breadth-first" (1)

```
void levelOrder(treePointer ptr){
   treePointer queue[MAX_QUEUE_SIZE];
   if (!ptr) return; /* empty tree
     */
   addq(ptr);
   for (;;) {
      ptr = deleteq();
      if (ptr) {
         printf("%c", ptr->data);
         if(ptr->leftChild)
            addq(ptr->leftChild);
         if (ptr->rightChild)
            addq(ptr->rightChild);
      } /* if */
      else break;
   } /* for (;;) */
}
```



→ +*E*D/CAB